LEGIBILITY NOTICE

LA-UR -87-3429 CONF-871007--36

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

LA-UR--87-3429

DE88 001832

TITLE SUBSYSTEM SOFTWARE FOR TSTA

AUTHOR(S) Lawry W. Mann, CTR-10
Garnett W. Claborn, CTR-10
Clair W. Nielson, CTR-10

SUBMITTED TO IEEE 12th Symposium on Fusion Engineering

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

SUBSYSTEM SOFTWARE FOR TSTA

Lawry V. Mann, Garnett V. Claborn, and Clair V. Nielson
Los Alamos National Laboratory
Los Alamos, NM 87545

Abstract: The software at the Tritium Systems Test Assembly (TSTA) is logically broken into two parts, the system support software and the subsystem software. The subsystem software controls the various physical subsystems at TSTA. Each physical subsystem is controlled by a single program. The program contains a concurrently running task for each device or group of similiar devices within a physical subsystem. The program interacts with the physical hardware through software library calls such as "OPEN", "CLOSE," and "SENSE." These library calls are provided by the system support software. The various tasks of a subsystem program communicate with the "main" task and the outside world through various global modes and status variables. By communicating in this highly structured way the possiblity for error is reduced. The logic of subsystem software is specified by a version of the Nassi-Schneiderman structured flowchart method and machine translated to executable code.

The Subsystem Control Software at the Tritium System Test Assembly (TSTA) must control sophisticated chemical processes through the physical operation of valves, motor controllers, gas sampling devices, thermocouples, pressure transducers, and similar devices. Such control software has to be capable of passing stringent quality assurance (QA) criteria to provide for the safe handling of significant amounts of tritium on a routine basis. Since many of the chemical processes and physical components are experimental, the control software has to be flexible enough to allow for trial/error learning curve, but still protect the environment and personnel from exposure to unsafe levels of radiation. The software at TSTA is implemented in several levels as described in a preceding paper in these proceedings. This paper depends on information given in the preceding paper for understanding. The top level is the Subsystem Control level.

Level Overview

The strategy in implementing the TSTA software in levels is to provide isolation between the functional parts of the software. The isolation allows for testing and modification of the various levels without undue impact on the other levels. Various levels communicate only through cleanly and rigorously defined interfaces. Each level performs specific tasks on specific inputs and outputs. (See diagram at end of paper.)

The lowest level is the hardware level. The function of the hardware level is to gather data from CAMAC modules and to command the CAMAC modules to various states. The data is collected by Front End Controllers and placed on an Ethernet link for reading by the next level. The next level, the Data Communications level reads data from the Ethernet link and constructs a database of raw (12 bit) CAMAC data. It accepts CAMAC execute commands from the next level as directives to pass to the hardware. The Data Communications level also does data archiving by storing a snapshot of the raw database onto the disk once a minute. The data is later compressed for long term storage. Logical variables which provide system status information are supported by this level. The Data Conversion level provides interfaces between the Data Communication level and the Operator Interface and Subsystem Control levels. At this level data is converted to and from engineering units and the raw database values. This level also provides limit checking and range information to the higher levels. The Operator Interface level provides the human operator with a window into the control system. Through this window the operator can both read the database values and set system parameters. The level also provides the Subsystem Control level with an interface for informing the operator of various subsystem information and for retrieving from the operator answers to specific subsystem software questions. The final level is the Subsystem Control level. The purpose of this paper is to describe the Subsystem Control level in some detail.

The Subsystem Control Level

The function of the software at this level is to control a specific subsystem of the TSTA fuel cycle. The main fuel cycle consists of several interconnected subsystems such as the FCU (Fuel Clean Up), the TWT (Tritium Waste Treatment), the TM (Tritium Monitoring), the ISS (Isotope Separation Subsystem) and others. The Subsystem Control level is organized along the lines of the physical hardware. This is advantageous because in general there is one non-software person (i.e. a physicist or chemical engineer) responsible for each physical subsystem. By restricting a program to dealing with one physical subsystem the software designer and subsystem designer interactions are simplified. Further, the physical subsystems interact with each other at very clearly defined boundaries; these boundaries became natural places to subdivide the software. Each physical subsystem is further subdivided into "components". For example the TWT contains as components the Low Pressure Receiver, the Compressors, the High Pressure Receiver, the Moisture Collectors and the Radiation monitors. The software that controls the TWT mirrors the physical decomposition into components. The operating system on the process computers allows multitasking with a single process. (On more restricted operating systems each task could become an autonomous process.) The software in each task controls one component of the subsystem. These tasks are grouped together to form a subsystem process, the program that is responsible for controlling an individual subsystem.

The communication between processes is highly restricted. The communication is done through logical mailboxes. When a process needs information about the status of another subsystem, it finds that information by reading one of the various status mailboxes belonging to that subsystem. These mailboxes are kept in the memory of the process computer and updated in the memory of the standby computer every 10 seconds. The mailbox contents are also written to the separate disks for archiving once a minute. Therefore the software of one subsystem does not directly interact with the software of another subsystem.

As an example, in the TWT process the process gas that is received may contain so much residual tritium that the TWT cannot adequately cleanup up the process stream in one pass. Rather than release tritium to the stack, the TWT goes into recycle mode. But in recycle mode the TWT would prefer not to receive any more gas to process. The gloveboxes which are under control of the TM program are nitrogen purging to the TWT. Now that the TWT is in recycle the purging should be stopped. There is a control for each glovebox, which if set, would inhibit the purging of the glovebox. Now the question is who should be responsible for the setting of this control? In a system where the interactions are not restricted the TWT might set the control. In such a system the number of programs that can access and control a single control point is variable. This makes testing the control algorithms difficult because of the (often unanticipated) interaction among subsystems. At TSTA the interaction is sufficiently restricted so that only one process can exercise control over any given control point. So when the TWT goes into recycle, it sets a status mailbox to that effect. The TM program monitors that mailbox and sets the glovebox to inhibit the purge. In this way the TM program can be tested without the presence of the TWT program.

During subsystem software testing the mailboxes can be set at the Operator Interface level to simulate the conditions that need to be tested. Another benefit of the process independence is the possibility of placing separate processes on separate computers in a multi or distributed processing environment. In a similar manner the interaction between subsystem tasks is restricted. This is of great importance when testing. Just as a subsystem process could be tested independent of other processes, individual tasks can be tested independent of other tasks. So testing occurs on a physical component basis. For each task (physical component) a software test plan is generated within the Quality Assurance (QA) program. When the test is executed the results are recorded in the appropriate QA file. After a subsystem has been QA tested, any changes made to the software must go through the QA department.

The subsystem software is written in structured flow chart specifications based on Nassi-Schneiderman (N-S) charts. These specifications generate graphical representations of the control algorithms that can be read by the subsystem designers who are not trained in any programming language. The actual input of the algorithms is done in a specialized language which generates both the structured flow chart and compilable code. By this means, no manual checking of the algorithm is necessary except in the format of the flow chart itself. In the present implementation, the code generation is through two steps, first to RATFOR, and then through the RATFOR preprocessor to compilable FORTRAN. This one-format representation of the algorithms is a major factor in the ease of maintaining and verifying software at TSTA.

The structured flow chart is constructed by the use of unique graphical representations for the standard programming constructs. Algorithms are specified through three such constructs, (1) the simple step, such as "SET TWT_CD_MSA1 OPEN", which means "open the path to the first molecular sieve bed". (2) the multiway branch, and (3) the iterative loop. By restricting the algorithm specification to three constructs, it is easier to learn to read and understand the algorithm specifications. The same text that generates the actual graphical representation is used to generate (under program control) the code used for the subsystem program. In addition to the flow chart specifications, each task has an English language description of the function of the task.

The simplest N-S chart to describe a subsystem process would be

```
Subsystem Process N-S Chart

 .-------------------.
 | control subsystem |
 '-------------------'
```

In the refining process it is noted that the subsystem is composed of various parts, for instance a compressor, a filter and a molecular sieve bed. So the chart then becomes

```
Subsystem Process N-S Chart

.------------------------.
| control filter         |
| control mole sieve bed |
| control compressor     |
'------------------------'
```

Now the filter, molecular sieve bed and compresser are not controlled just once, but on a cyclic basis.

```
Subsystem Process N-S Chart

.----------------------------------.
| While SUBSYSTEM.MODE = RUN       |
|   .------------------------------|
|   | control filter               |
|   | control mole sieve bed       |
|   | control compressor           |
'----------------------------------'
```

This gives an example of an iterative loop. Adding more detail to the compressor control results in.

```
Subsystem Process N-S Chart

.------------------------------------------------------------.
| While SUBSYSTEM.MODE = RUN                                 |
|   .--------------------------------------------------------|
|   | control filter                                         |
|   | control mole sieve                                     |
|   |--------------------------------------------------------|
|   | WHAT IS compressor shell temperature ?                 |
|   | > 1000                | > 500              | Other     |
|   |-----------------------|--------------------|           |
|   | turn compressor off   | warn operator      |           |
|   | warn operator         |                    |           |
|   |--------------------------------------------------------|
|   | further compressor control                             |
'------------------------------------------------------------'
```

The new structure is a multiway branch. If the compressor temperature is greater than 1000 then it is also greater than 500. The second branch is not taken because the multiway branch takes the first true branch. If the temperature is greater than 500 and less than or equal to 1000, then the second branch will be taken. The "other" branch is always true and will be taken if the temperature is less than or equal to 500.

At some point in the refinement of the subsystem the N-S chart will grow too large to fit on one page. At this point a decision will have to be made as to where to divide the N-S chart. A logical place to divide the N-S chart would be to place the compressor control on a separate page. This shows how the N-S charts expand through magnification, not by line connection.

```
Subsystem Process N-S Chart

.----------------------------------------.
| While SUBSYSTEM.MODE = RUN             |
|   .------------------------------------|
|   | control filter                     |
|   | control mole sieve bed             |
|   | execute "compressor control"       |
'----------------------------------------'
```

Compressor control N-S Chart

WHAT IS compressor shell temperature ?		
> 1000	> 500	Other
turn compressor off warn operator	warn operator	
further compressor control		

The N-S Charts become the focal point of the software QA. Changes to the subsystem software is described in terms of N-S charts. All such changes are passed before a review board prior to implementation.

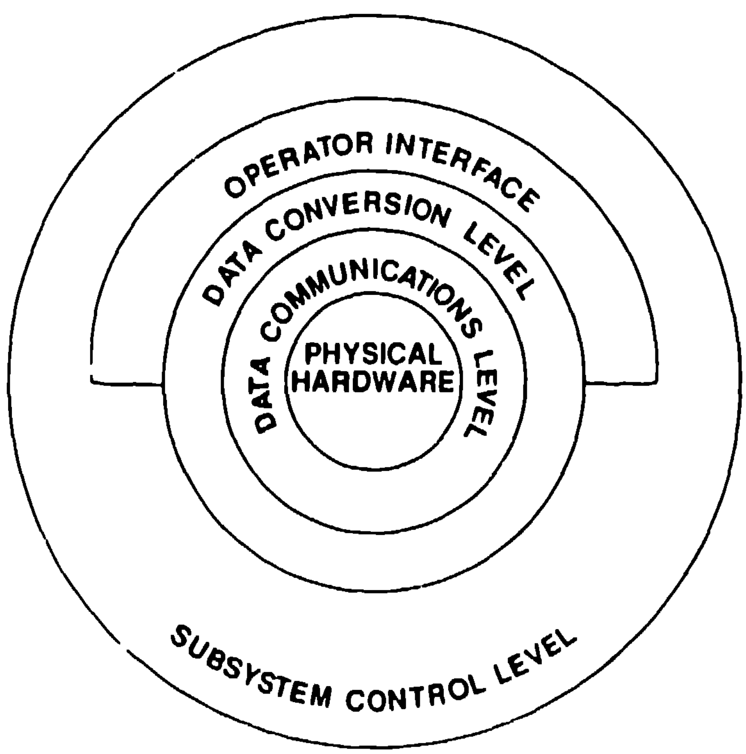
The Subsystem Control level interacts with the lower levels in a very structured manner. For example, all TSTA parameters are specified in the subsystem control software as symbolic constants like "TWT_P_LPR1" (the reading of the pressure in the TWT low pressure receiver). The subsystem interacts with the Data Conversion level through "SENSE" and "SET" commands. The details of how the information is gathered from the CAMAC modules is hidden from the Subsystem Control level. By hiding the details of how TSTA variables are read and set, the subsystem control software can be written and verified more easily.

The Subsystem Control level interacts with the human operator through the Operator Interface level. The operator can be informed of equipment failure (such as a valve that failed to close), and of subsystem parameters and status. The operator interacts with the subsystem software by the setting of various logical mailboxes that the subsystem reads to determine which mode to run in, or what path to take, or what components should be put offline for maintenance. Because all subsystem parameters are kept in mailboxes and the Data Communication level is keeping both process computer's mailbox database current, computer switchover is painless. Each level is brought up and the Subsystem Control level is started last. When started it finds all the pertinent data in the mailbox database and maintains the same subsystem state as was on the other process computer. The physical components are not perturbated by the computer switchover. The switchover process takes about 2 minutes.

Conclusions

The advantages of implementing the TSTA control software in various levels has been stated. Implementing the Subsystem Control level in such a way as to isolate each subsystem process has several advantages. It provides a much needed modularization to keep any given program to a manageable size. Since physical components can only be controlled from one subsystem process, access control to the physical devices is readily implemented. Subsystem software debugging and verification of a particular subsystem can be done independently of other subsystems. By keeping the subsystem interaction to a minimal amount, the amount of testing needed to verify correct subsystem integration is reduced. By further reducing a subsystem process software into autonomous tasks, the process of control algorithm specification and testing is made easier.

TSTA SOFTWARE LEVELS



COMMUNICATION CAN OCCUR
ONLY BETWEEN ADJACENT LEVELS